Technical Report

CMU/SEI-89-TR-026
ESD-TR-89-034

CASE Planning and the Software Process

Watts S. Humphrey

May 1989

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAY 1989		**00-00-1989 to 00-00-1989**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CASE Planning and the Software Process	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Software Engineering Institute,Carnegie Mellon University,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
http://www.sei.cmu.edu/pub/documents/89.reports/pdf/tr26.89.pdf

14. ABSTRACT

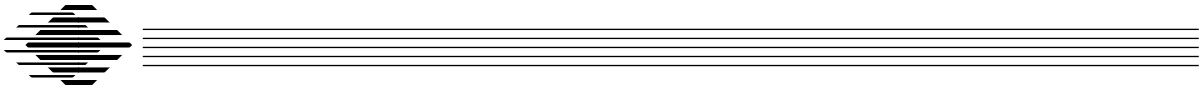
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Technical Report

CMU/SEI-89-TR-026
ESD-TR-89-034
May 1989

CASE Planning and the Software Process



Watts S. Humphrey

Software Process Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through SAIC/ASSET: 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 / FAX: (304) 284-9001 / World Wide Web: http://www.asset.com/sei.html / e-mail: webmaster@www.asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

List of Figures

CASE Planning and the Software Process

Abstract: Automating a software process both magnifies its strengths and accentuates its weaknesses. Automation can make an effective process more effective, but it can make a chaotic process even worse—and at considerable expense. Anyone who buys expensive tools to solve an ill-defined problem is likely to be disappointed. Unless procuring such tools is part of a thoughtful software process improvement plan, the purchase could be an expensive mistake.

This report discusses software process maturity and its relationship to planning and installing computer-aided software engineering (CASE) systems. While process is not a magic answer (there isn't one), the key issues are discussed from a process perspective, and guidelines are given for avoiding the most common pitfalls. Since CASE systems can involve significant investment, an economic justification may be necessary. The relevant financial considerations are therefore discussed, and some basic steps for producing such justifications are outlined. Finally, some key considerations for introducing and using CASE systems are discussed.

1. Introduction

The reason we use computer-aided software engineering (CASE) systems to automate the software process is to improve the quality and productivity of our work. When we can reduce a task to a routine procedure and mechanize it, we both save labor and eliminate a source of human error. The latter, it turns out, is the most effective way to improve productivity: greater software productivity improvements come from eliminating mistakes than from performing tasks more efficiently [HUM89b]. While better tools and procedures cannot replace creative designers, poor ones can frustrate and impede them.

To do a consistently effective job on a complex task, we need to consciously understand what we are doing. While some people are so talented that they unconsciously do superb work, an intuitive process is more risky with larger tasks and multi-person teams. A more orderly focus on process definition and improvement is then advisable.

For a complex process to be understood, it must be relatively stable, not subject to wild fluctuation. Deming refers to this as statistical control [DEM82]. When a process is under statistical control, repeating the work in roughly the same way will produce roughly the same result. Then, to get consistently better results, we must improve the process. Sustained improvement is possible only if the process is under statistical control. Thus, the first management challenge is to make the software process reasonably stable and effective. As Stenning says, "the role of an environment is to support effective use of an effective process [STE87]." After the process has become stable and is reasonably effective, we can consider automation to improve its efficiency.

Before proceeding further, it is necessary to define what I mean by CASE. One published definition says that "computer-aided software engineering (CASE) is the application of automated technologies to the software engineering procedures" [CAS86]. This definition encompasses the entire range from individual task-oriented tools to fully integrated operational software environments. My discussion in this paper addresses the latter: planning and installing a comprehensive software environment. Such an environment should include a set of compatible tools, a common database, task management facilities, and provisions for configuration control. When I use the term "CASE system" in this paper, I am referring to this type of comprehensive support environment.

While some CASE systems presume design and implementation methodologies, others provide more flexibility—and consequently less guidance. As CASE systems become more sophisticated, they are necessarily more closely involved with the practitioners' personal working practices. As a result, common disciplines and methods become necessary, along with common procedures, conventions, standards, interfaces, and measures. If an organization has not already established such a common working framework, the installation of a CASE system can be traumatic. Even though a common framework is a key aspect of a mature software process, the constraints could easily seem to result from the CASE system itself. Such apparent sharp reductions in the freedom and flexibility of the working professionals can cause resentment and resistance unless the system is carefully planned and introduced.

This paper also discusses CASE systems in a somewhat idealized sense. While the full range of capabilities is not yet entirely clear, it is likely that CASE systems will evolve in much the same way as many of the other system facilities common to our field. A large-scale database system, for example, provides a basic framework and a set of facilities. The users then decide how to use these capabilities. They must define the data types to be handled, their precise formats and relationships, the desired outputs and their formats, and the operational and control procedures. The design of the database system imposes constraints on many of these decisions, but the users generally have considerable flexibility in their precise use.

In the comprehensive CASE systems of the future, the situation should be similar: a wide range of basic capabilities will be provided, but the users will have to select the systems which most closely meet their needs and then tailor the specific protocols, formats, procedures, and methods to their precise tasks. For relatively standardized uses, it is also likely that sets of "canned" facilities will be provided so that less demanding users could accept these built-in choices rather than tailor their own. This approach, however, also poses a serious danger to the uninformed: the standardized CASE system facilities may not fit the user's intended need. This is why, at least at this early stage in CASE systems development, uninformed users must be doubly careful. They should first decide on the methods that best fit their needs and only then select a CASE system to support these methods.

2. Software Process Maturity

In early 1987, the Software Engineering Institute (SEI) at Carnegie Mellon University launched a program dedicated to working on the software process. The objective was to help software organizations to improve their software capabilities.

The initial work of the program was based on a request from the U. S. Air Force for a method to help identify the most capable software suppliers. SEI staff developed such a method, which has been successfully used on a trial basis by the Army, Navy, and Air Force to assist in their software source selections [HUM87b]. Additional work is underway to refine the method and make it more broadly available to U. S. government agencies.

The focus on software process is based on the premises that (1) the process of producing and evolving software products can be defined, managed, measured, and progressively improved and (2) the quality of a software product is largely governed by the quality of the process used to create, repair, and improve it.

The software process is the set of activities, methods, and practices which guide people in the production of software. An effective process must consider the required tasks, the tools and methods, and the software developers' skills, training, and motivation.

Successful software work requires capable and motivated technical people, knowledge of the ultimate application environment, and detailed understanding of the end user's needs [CUR88]. These alone, however, are not enough: the key missing ingredient, generally, is effective software process management. Software process management is the application of process engineering concepts, techniques, and practices to monitor, control, and improve the software process. As software organizations grow larger and their projects become more complex, process management becomes progressively more important. In fact, inattention to this one topic causes many otherwise capable software groups to be ineffective.

To address software process issues, the SEI developed a process maturity model and a related software process assessment instrument. These are the basic elements of our methods for examining and improving the software process [HUM87a]. Using these methods, an organization's software engineering capability can be characterized with the aid of the software process maturity model. By using the SEI method, organizations can determine the priority actions required for improving their software capability.

As shown in Figure 2-1, this model consists of five maturity levels and the key activities required at each level. These maturity levels have been selected because they:

- Reasonably represent the historical improvement phases of actual software organizations.
- Represent a measure of improvement that can realistically be achieved from the prior level.
- Suggest interim improvement goals and progress measures.
- Make obvious a set of immediate improvement priorities once an organization's status in this framework is known.

Figure 2-1: Software Process Maturity Model

At the initial level (level 1), an organization can be characterized as having an ad hoc, or possibly chaotic, process. Typically, the organization operates without formalized procedures, cost estimates, or project plans. Even if formal project control procedures exist, there are inadequate management mechanisms to ensure they are followed. Tools are not well integrated with the process, nor are they uniformly applied. Change control is rarely adequate, and senior management is not exposed to or does not understand the key software problems and issues. When projects do succeed, it is generally due to the heroic efforts of a dedicated team rather than the capability of the organization.

An organization at the repeatable level (level 2) has established basic project controls: project planning, management oversight, software quality assurance, and change control. The strength of the organization stems from its experience at doing the same kind of work many times, and it generally does reasonably well at meeting its schedule and cost commitments. Such organizations, however, face major risks when presented with new challenges. They also have frequent quality problems and generally lack an orderly framework for process improvement.

At the defined level (level 3), the organization has laid the foundation for examining the process and deciding how to improve it. The keys to moving from the repeatable level to the defined level are to establish a software engineering process group (SEPG), to establish a software process architecture that defines the key process activities, to review the adequacy of the training program, and to introduce a family of software engineering methods and technologies. Although all these issues could usefully be addressed at level 2 or even at level 1, special focus on them is required at this point to permit further progress to level 4.

The managed level (level 4) builds on the foundation established at the defined level. When the process is defined, it can be examined and improved, but there is little data to measure effectiveness. Thus, to advance to the managed level, an organization needs to establish a minimum set of measurements for the quality and productivity parameters of each key task. The organization also needs a process database with resources to manage and maintain it, to analyze the data, and to advise project members on its meaning and use. This data is useful not only for managing the process, but also for justifying the necessary investments in training, staffing, and CASE systems.

Two requirements are fundamental to advance from the managed to the optimizing level (level 5): data gathering should be automated, and management should give highest priority to process analysis and improvement. At the optimizing level, the organization has the means to identify the weakest process elements and strengthen them; data are available to justify applying technology to the most critical tasks; and numerical evidence is available on process effectiveness. The key additional activities at the optimizing level are rigorous defect cause analysis and defect prevention, which are so important because they provide a routine mechanism for maintaining focus on process improvement. When each key task is completed, the people who were involved conduct a brief postmortem to determine what went wrong, what should be fixed, and what could be improved. This provides the organization with the essential data for making steady process improvements.

While there are many aspects to the transition from one maturity level to another, the basic objective is to achieve a controlled and measured process as the foundation for continued improvement.

This maturity framework has been used as the basis for an assessment program conducted by the SEI to determine the state of the software process in United States software organizations. From this work, it appears that approximately 84% of the organizations are at level 1, and only about 14% are at level 2 [HUM89a]. It is thus likely that most organizations considering the installation of CASE systems will be at level 1, the least mature level of software process management. Before addressing the key issues these organizations are likely to face, I will discuss the reasons for automating the software process.

3. Why CASE Is Needed

At present, software development is a human-intensive process with all the limitations this implies. For example, people can do things only so fast. We accept the four-minute mile as near the limit of human capability and see the three-minute mile as beyond human aspiration. Our intellectual capabilities are also limited. Airline pilots, for example, can only focus on a limited number of actions at once and their performance degrades with fatigue. There are similar limits to the quality and production of software developers, but they are not as clear. While human capabilities do seem to improve with time, these improvements are only gradual. Clearly, to make major improvement, our hopes must increasingly rest on some new conceptual view of our tasks or the use of more powerful tools.

While we can and should continue our search for better concepts and methods, consistent and sustained software process improvements must ultimately come from technology. Improved tools and methods have been helpful throughout the history of software; but once the software process reaches level 5 maturity (optimizing), we will be in a far better position to understand where and how technology can help.

In advancing from the level 1 (chaotic) process, we can make major improvements by simply turning a crowd of programmers into a coordinated team of professionals. Perlis put it best in 1968 when he said, "We kid ourselves if we believe that software systems can only be designed and built by a small number of people. If we adopt that view this subject will remain precisely as it is today, and will ultimately die. We must learn how to build software systems with hundreds, possibly thousands of people [PER68]." This is the challenge faced by immature software organizations: how to plan and coordinate the creative work of their professionals so they support rather than interfere with each other.

Software process management provides a framework for progressively improving the orderliness of our work. By moving from the initial (level 1) to the repeatable (level 2) and then to the defined (level 3) process, we establish the commitment framework needed to coordinate large operations. By using a structured, managed, and planned process, our professionals better understand their roles and their interrelationships. The defined software process then facilitates more rational reactions to the surprises and challenges of our work.

Once orderly performance is achieved, improvement continues but only in increments. At some point, the professionals are using the process about as effectively as they can; yet we will still need further quality and productivity improvements. At this point, what additional steps can we take, and what limitations will we ultimately face?

While automation of our current process may help, automation alone is unlikely to get us very far. We must couple automation with improved concepts for defining and coordinating our work and for designing and validating our work products. Through the power of automated environments, we will better relate to and capitalize on the work of others. The need is for better characterizations of software objects, better processes for controlling and relating them, and more powerful ways to access and manipulate them. This combination of

method, process, and automation will, hopefully, enable software professionals to start down the same path that has fueled four centuries of scientific progress. While software reuse has long been more a hope than a promise, the greatest opportunity for dramatic productivity and quality improvement is likely to be through improved ways to build new software on the progressively richer foundation of previously produced products. We will then, to paraphrase Isaac Newton, build our products on the achievements of our predecessors. Making this dream a reality will require a creative combination of method, process, and technology.

There is also the potential for significantly improving software process support. By making our tools more consistent and integrating them in a common CASE system, we will move beyond task support to support of the full life cycle. As CASE systems grow more sophisticated, we can expect substantial improvement in their ability to support common understanding of management and technical plans and in their ability to guide the execution of individual tasks so they each better relate to the total job. Any large software project is performing a complex process, and it is often difficult to know which steps are most appropriate at any point. Data gathering and analysis also require support in order to be more accurate, timely, and economical. While automated support for these activities will come only gradually, organizations should take early steps to develop a software process support strategy. When this is coupled with a concerted effort to improve the process maturity to level 3 or better, the organization will have a sound framework for orderly and steady improvement.

4. CASE Planning Guidelines

While it is risky to propose universal guidelines at this early stage in CASE technology development, the following five appear to be generally applicable. Clearly, with further experience we will learn a great deal more. While I believe these CASE planning guidelines will be reasonably durable, they are based on limited experience. They must, therefore, be reexamined as new data becomes available.

4.1. Guideline 1 - If your process is chaotic, get it under control before attempting to install a CASE system.

A level 1 organization must get its software process under control before it attempts to install a sophisticated CASE system. In short, its first priority is to reach level 2. To be fully effective, CASE systems must generally enforce standard policies and procedures on their users. If the systems are used with an immature process, the environment will merely become another bureaucracy, only an unavoidable one. The saving grace of most immature organizations is that their poor management system permits their ineffective process to be circumvented by their professionals. The truly unworkable steps have been skillfully evaded, thus permitting the professionals to produce in spite of the system. When a CASE system enforces an unworkable process, it generally brings the level 1 organization to its knees.

It is possible to use CASE systems as collections of individual tools. Although this is an expensive way to obtain compilers and editors, it can be a reasonable step toward standardizing on a single tool set in preparation for later support improvements. For any more sophisticated use, however, the most important single rule is: if your process is chaotic, get it under control before attempting to install a CASE system.

4.2. Guideline 2 - Develop your process before or during CASE installation, but not after.

Many organizations that work with CASE systems have barely reached level 2 or are struggling to get there. Thus, they have not made much progress in defining their process. The first task such groups generally formalize is configuration management, and some groups may even establish strong procedures for controlling requirements. These are appropriate processes to consider for initial trial implementation with a CASE system.

Beyond this, the organization should establish a full-time group of experienced software development professionals to work on process improvement. This group should either include or work closely with the group responsible for planning and supporting CASE implementation. These two groups, the software engineering process group (SEPG) and the technology focal point, then form the core for continued software process development and automation. If such dedicated groups are not established, the process will likely stagnate

and the CASE system will cease to evolve in step with the organization's needs and capabilities. When no one is assigned to work on process development, improvement is unlikely—after all, the process can't improve itself.

4.3. Guideline 3 - System conversion is critical, but converting the people will be the hardest job of all.

While there are many facets to the "people issue," training is one of the most important; and it is most often overlooked. In fact, training is the most common single weakness of level 2 organizations. Professional programmers often will not accept and may not even be able to use advanced tools effectively unless they are adequately trained.

Training is also very expensive, and there are many different groups that each have their unique training needs. In one organization, just the tool training for a 30-person programming shop was $162,000, and this did not include the required instructional hardware and software. In another case, per-tool training costs were projected as $1,507 per person, so each professional's training cost for a simple six-tool CASE system was $9,042 [HUG88]. Beyond this, we must consider training in languages, design methods, inspections, and testing, to name just a few of the topics. Training is likely to be very expensive, but not nearly as expensive as not training.

4.4. Guideline 4 - Recognize that a CASE installation is never completed.

The people, their experience level, the available technology, process methods, the business environment, and project needs all change with time. The environmental support system must thus also change and evolve. While it is essential to maintain reasonable environmental stability, people's capabilities change and new problems arise which require attention. It is therefore essential to maintain continuing resources to plan, support, and control CASE evolution. This requires a permanent staff, a system for reporting and tracking problems, mechanisms to periodically assess the users' needs, and the capability to test and prototype changes before putting them into general use.

As pointed out in guideline #2, a technology focal point is needed to provide assistance, interface with the vendors, and lead the CASE planning and installation efforts. These tasks, however, require a delicate balance between the roles of champion, advocate, and manager. The champion sells the organization on the need for CASE. The advocate understands the organization's needs and represents them to the vendors. The manager develops the plans, musters the resources, and directs the implementation. All this, of course, implies an orderly, planned approach. That is another reason for guideline #1: if the organization's software process is chaotic, its CASE planning and installation efforts will likely be chaotic as well.

4.5. Guideline 5 - Don’t forget to think!

With inappropriate automation, it is easy to induce online frenzy, abstraction distraction, or object hypnosis. The introduction of powerful tools carries the risk of damage through misuse. For example, powerful tools make mindless action so easy that enormous heaps of code can be produced without much thought. Although it is generally possible to use a CASE system as an aid to a manual architectural design, both managers and implementors get impatient waiting for design completion and may prematurely rush into implementation. When the environment’s powerful facilities have been oversold, management may believe it is capable of far more than it can possibly provide. Good designs start with functional and structural concepts, and no machine can produce these. When the developers rapidly produce mountains of detail, it is easy to believe that the environment has provided an orderly framework. Without clear concepts, this frenzied rush to implement can easily produce abstraction nestings with dozens of levels or case statements with thousands of choices. When such absurdities pervade a system, it can become irreparable even with the best available tools.

The basic guideline is: select your design methods with care, and not just because they came with the tool; then insist on a thorough high-level architectural design. This will provide a solid foundation for everything that follows.

5. Planning and Installing CASE Systems

Although a full treatment of CASE planning and installation is beyond the scope of this report, a few general considerations are worth special attention.

First, just as with a product effort, it is advisable to start with an orderly study of the requirements. The requirements clearly state management's objectives for the CASE system and define the critical functions. These functions should be prioritized, however, or they will likely become so voluminous as to provide little guidance. After all, if everything is needed, anything will fall within the intended scope. Since not all desired functions can be provided at once, all-encompassing requirements merely leave the task of selecting priorities to the implementors.

One approach is to establish three priorities: A - essential for the first installation; B - required as an early enhancement; and C - desirable in the future. It is wise to get both technical and project management agreement with these functional priorities. As the planning becomes more specific, installation and conversion commitments will be facilitated if the A, B, and C priorities are known for each product area. If no product area is willing to convert to and use the CASE system at an early date, the plan should be reevaluated and the A priority items reestablished.

Over time, comprehensive CASE systems are likely to be the single most expensive capital investment a software organization makes. Although it is not clear what such systems will cost, I believe that ultimately CASE system costs will exceed those of the organization's other software, computing systems, and terminals combined.

Most software professionals can get approval to purchase a software package for a few hundred dollars. While you may have to answer some financial questions, a little arm waving and technical mumbo jumbo will generally suffice. When the costs mount to a few thousand dollars, a manager's approval is required; but, again, as long as he or she agrees to support you, arm waving will generally win the day. When the costs rise much beyond this, hard data will be required. The question then becomes: what data and how can it be presented to win approval?

There are several arguments for getting large expenditures approved in dollar-conscious industrial organizations:

1. A recognized outside expert provides testimony. This is not a very reliable method and will not alone suffice when the costs reach a million dollars or more. Since it will be some time before CASE system costs reach that financial stratosphere, the expert route will likely be adequate for the near term.
2. The leading competitor has one. This is almost foolproof. Unfortunately, it requires a competitor who is willing and able to be first. At least for that competitor, this returns us to the previously unsolved problem.
3. The contract requires it. Here, the odds favor agreement to get a CASE system. If history with some early software tools is any guide, however, this sys-

tem will be provided as GFE (government furnished equipment) and will not be ready at contract start time. Unfortunately, this has historically meant that a reliable and fully functional CASE system will not even be ready at contract completion.

4. There is economic justification. Developing the economic justification requires work, but it can be done. Some suggestions on how to do this are given in the next section.

5.1. Developing the Economic Justification

Although any reasonably conducted financial study will likely produce the required results, significant work is required. The following steps are not proposed as the only or even the best way to produce such a justification; however, they have been used with some success.

1. Develop a task map for the technology currently in use and another task map for that which is planned.
2. Identify the current resource expenditure profiles for each task. These numbers must be reviewed and accepted by the leading project managers.
3. Assemble a team of experts to determine the resource impact of the new CASE system and its associated support tools and methods. This team should also consider the likely quality consequences at each process stage and should use the same level of detail as the task maps. When completed, this information is also reviewed with the project managers and their leading technical people.
4. Review each project's migration plan to determine the accrual rate of the savings. Don't forget to include a learning curve as well as the anticipated product quality benefits in the test, installation, and repair phases.
5. Get each project's commitment to this savings schedule, together with its agreement to adjust project plans accordingly. This will not be easy, but it is the only way to convince the finance people that this investment is worthwhile.
6. Establish a new schedule of estimating factors for project planning.
7. Estimate the planning, training, installation, conversion, and support costs.
8. Construct a composite savings schedule for the organization as a whole. If the information is available, reference to other organizations' experiences can add considerable credibility.
9. Get competent financial advice on the methods used and the results anticipated.
10. With this justification, request management approval for the plan.

While most of these steps are relatively straightforward, the task map in step 1 is not. Essentially, it consists of a structured picture of the software process with identification of each key task and the tools and methods used in performing it. The work is analyzed in sufficient detail to identify the specific costs associated with each of these key tasks and to determine the anticipated costs with the new CASE system. Because some of the most significant savings are likely to result from improved quality and the resulting reduced testing and repair costs, it is also essential to estimate the improved quality consequences for each process task.

The development of an economic justification is impossible without the support of project managers. Even with their support, it will be hard to sell cost-conscious senior managers on a major investment that does not directly increase revenue. With enough support and persistence, however, the case can be made.

5.2. Economic Justification - Special Considerations

In generating an economic justification, it is helpful to keep these special considerations in mind:

1. Level 1 organizations don't have the data to make an economic justification. Level 2 organizations generally have to make an extensive study to produce a rudimentary justification. Level 3 groups have the cost data but need to conduct a special study to estimate the quality factors. Level 4 and 5 groups have the needed data readily at hand.
2. In projecting savings, don't get too far out on a technological limb; focus on practical and achievable improvements.
3. Involve the financial people. A professionally prepared financial story takes a lot of work, but it will be far more convincing to the intended audience than a technical presentation.
4. Do the work in detail. Though the detail will not be presented, bring a well-structured summary of the results as backup. It is rarely needed, but a thick pile of backup material builds the presenter's confidence and deters the most detail-minded financial executive.
5. Approval is unlikely unless you have line management's agreement and savings commitments from the project managers.
6. Remember that there is no way to absolutely prove the economic value of a CASE system. This, however, is generally true of any other capital investment. Although this is not a useful arguing point, it is worth remembering during the financial debates. When the executives are convinced of the soundness of your logic, they will not require absolute proof.
7. It takes time to do it right, but there is no shorter way.

One thing to remember: be very careful about promising dramatic early savings. There is evidence that a fully automated CASE system, by enforcing a comprehensive design discipline, will actually increase early costs [LEM88]. While these costs are the natural result of enforcing an orderly and complete requirements and design process, the CASE system could be blamed and the entire effort jeopardized. Since such cost increases generally result from more complete early design work, the later benefits in reduced testing and improved product quality could easily pay for the entire CASE installation.

The early costs are also likely to be inflated by the added time required for the professionals to become familiar with the new system and the procedures and methods involved in its use. This learning curve effect should be tracked with the early installations so that subsequent users are better informed when making their plans.

5.3. Installing CASE Systems

If the planning has been done properly and adequate training and support facilities are available, the actual installation and conversion to a CASE system should be relatively straightforward. This does not mean that this phase is easy, however, because any change in the working fabric of an organization is a major upheaval, and many surprises and problems will invariably occur. With adequate preparation, these problems can generally be handled.

One possible exception is an attempt to convert an existing project to a new CASE system. This is always difficult, but it should not even be attempted unless the project is at or very near level 2. Such projects generally have larger staffs as well as major investments in existing designs, test suites, and code. They should, therefore, treat claims for improved quality and productivity with some skepticism and carefully balance the promised benefits against the more certain costs and risks of a change.

On the other hand, if the project is near the beginning of a long development program or is early in its maintenance phase, there may be sufficient reason for a change. The long-term benefits of improved tracking, better design disciplines, and comprehensive change control are often sufficiently attractive to warrant conversion. The key rule, however, is to always keep one foot on solid ground. This requires extended dual system operation and successful performance in slave mode before the first trial cut-over. This approach can be very expensive and, in some cases, may not even be feasible. The costs of unanticipated failure are so significant, however, that any precipitous cut-over would be foolhardy. Again, such moves should be attempted only if the organization is at or very near level 2.

6. Designing with CASE

Perhaps the most important design consideration with CASE systems is that so little is different: the old familiar design principles are still essential. The combination of an effective process and a powerful CASE system can be enormously helpful, but it cannot do everything. When the requirements are wrong, when the application is not understood, or when the conceptual design is ill-founded, product results are invariably poor, regardless of the process, tools, or methods used. Though a well-defined and managed process can help identify these problems, it cannot prevent them.

A mature process can and should ensure that the proper time and resources are devoted to requirements development, that application needs are thoroughly studied, and that adequate design resources are invested early in the project. It can also ensure a competent expert review of the design itself. When management then attends to the identified concerns, a sound foundation for product development will likely result. Under these conditions, an automated environment can be of greatest value.

With a less mature process, requirements resolution is typically ad hoc; little conscious attention is paid to the design; and there is little provision for expert technical review. Often, customer participation in some technically superficial design review is taken as sufficient evidence that this early work has been done and that code production can begin. And often the result is either a poorly designed product or an expensive and time-consuming remedial design effort during test.

Generally these problems are not greatly affected by the introduction of CASE systems; but the lack of a coherent and well thought out conceptual design can cause special problems. Here, the combination of an immature process and an automated environment can create a false sense of security. The problem is that the CASE system depends on the product's structural design for the partitioning of work between the subsystems, components, and modules. When this system structure lacks coherence, the entire project organization and its support system will be similarly flawed. If this problem is recognized in time and a remedial design effort is launched, the CASE system can greatly assist in the recovery. More often, however, these problems are not recognized until product integration; and then it is hard to stop and rethink the design. More often, management will push ahead and try to patch the problems as they arise. While this may produce something that actually works, the underlying structural flaws will likely result in a system that is difficult to integrate, impossibly slow, and a maintenance nightmare.

The problems of an incoherent design are always severe, and a CASE system can make them worse. The apparently orderly work flow and the quiet efficiency of change management and status reporting lull the management into believing the product really is well structured. This confusion of efficient method with sound concept can have painful consequences. Designs may have multi-layered abstractions with a dozen or more levels or an amoeba-like sea of functional objects with no coherent relationships. When such problems pervade the fabric of a system, performance degradations of 1000 times or more are likely,

and the resulting kludge may be more expensive to fix than scrapping the design and starting over.

6.1. People Considerations with CASE

Process maturity can be viewed as an evolution in the way people work. At the lowest level (level 1), we have a largely uncoordinated collection of individuals. Though they may occasionally work together cooperatively, this is more incidental than a result of organization or process. At the next level (level 2), we have collections of small professional teams who work well as single groups but do not coordinate effectively, except on a largely personal basis. Finally (at level 3 and above), we have larger scale organizational frameworks within which many teams cooperatively interact and support each other. Process maturity is thus not just a management issue but one of establishing the human environment for a large-scale cooperative endeavor.

Perhaps the greatest single people issue is the need to build a common working ethic and practice. It takes work to build close-knit teams, and many software people object to standard procedures. Enabling a crowd of individual programmers to become a coherent team is a key result of moving from maturity level 1 to level 2. This evolution is a prerequisite to the adoption of the common methods and tools of a CASE system. In spite of the most elaborate preparation, if the professionals are not ready to accept the CASE system as a common working framework, they will treat it as another constraint to be evaded. They have painfully learned to publicly accept imposed policies and procedures and then to do whatever is needed to get the job done. This is rarely in accord with the official process. Without a thoughtful program to win the professionals' allegiance, a CASE system installation can be an expensive blunder.

The current vague process definitions typical of most software organizations give the professionals considerable freedom. Stenning points out that "most of the processes used in the software industry would be completely unworkable were it not for human ingenuity and flexibility [STE87]." In implementing CASE systems, we must recognize that no one is smart enough to define every precise detail of even relatively small parts of the software process. We must be careful that the imposition of constraints and controls always leaves room for exceptions and escapes. Strong facilities are needed for such items as protection, recovery, change authorization, and promotion control; but even these will need management-authorized escapes. Within these fundamental constraints, the system should permit wide latitude on what tasks are performed, when, and how. As process maturity improves, the professionals better understand how each task should generally be performed and are better able to judge when and why to use alternatives or exceptions.

6.2. Software Teams and CASE

When a small, coherent team embraces a common process and the CASE system which supports it, powerful results can be expected. When the team retains its structure and has control of its own pace for adopting new practices, its members will likely retain their cohesiveness even through the installation of a new support system. Such teams are most likely to make a successful CASE installation.

An important reason for their success is that the CASE system installation legitimizes their efforts to define and improve their working process. We readily accept a football team's need to plan and practice its plays, but somehow the similar needs of programming teams are not recognized. By making this work part of the CASE system migration effort, it becomes more acceptable. Such process work can actually improve a team's performance without the installation or use of a single tool. By ensuring that the entire team fully understands the methods that each member has found most effective, the performance of most teams will improve significantly.

6.3. The CASE Bureaucracy

Even though it will not be apparent until some time after a CASE system has been installed, there is an interesting converse to the acceptance problem. Over time, some professionals will even believe that the CASE system can do no wrong. They will then accept what it produces and blindly follow the steps and actions it prescribes. At that point, the CASE system will have become another and more pervasive bureaucracy which can block rational thought. Though the operation may seem more efficient, its procedures and methods will gradually become less pertinent to current problems. The resulting petrified process can then actually retard organizational improvement. Actions must be taken to ensure that the professionals continue to "own" their own software process, or they will not evolve it in concert with their needs and capabilities. This, of course, is the role of the level 5 optimizing process: it provides the people with a mechanism to review and modify the process in accordance with their current needs.

7. Conclusion

CASE systems hold enormous potential if properly used, but they can also be costly mistakes. When such systems are installed to solve vague and ill-defined problems, they will be a serious disappointment and may even cause harm. After your organization has its software process under control, however, you should certainly investigate CASE systems.

Once you decide to proceed, I suggest you consider the following initial steps [HUM89b]:

1. Assess your organization. Conduct an orderly study, determine the key problems, and initiate appropriate improvement actions.
2. Establish a software engineering process group (SEPG) to focus on and lead process improvement. If no one is working on process development, improvement is unlikely, for the process can't improve itself.
3. Establish an automation focal point. This responsibility should be assigned to someone who is part of or closely associated with the SEPG and is capable of leading the CASE planning and implementation work. Though such an assignment may not be needed immediately, it will be appropriate as soon as the organization approaches process maturity level 2; and it is absolutely essential to initiating work on installing a CASE system.
4. Plan, manage, and track process improvement with the same discipline you use to plan, manage, and track the projects.

Acknowledgements

In preparing this report, I have been fortunate in the quality and depth of advice and review I have received. I am particularly indebted to Brett Bachman, Grady Booch, Anita Carleton, Ken Dymond, Peter Feiler, Louise Hawthorne, Bob Kirkpatrick, Dave Kitson, Steve Masters, Don O'Neil, Mark Paulk, Ron Radice, Tom Rappath, and Cindy Wise for their helpful comments and suggestions. Linda Pesante has provided very professional editorial assistance, for which I am also most grateful. Any errors, omissions, or misstatements are, of course, entirely my responsibility.

References

[CAS86] Case, Albert F., Jr., *Information Systems Development: Principles of Computer-Aided Software Engineering.* Englewood Cliffs, N.J.: Prentice-Hall, 1986.

[CUR88] Curtis, B., H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large systems," *Communications of the ACM,* November 1988.

[DEM82] Deming, W. E., *Out of the Crisis.* Cambridge, Mass.: MIT Center of Advanced Engineering Study, 1982.

[HUG88] Hughes, A., "The Care and Fielding of a Software Development Environment," *Transferring Software Engineering Tool Technology,* S. Przybylinski and P. J. Fowler, Eds., IEEE Computer Society Press, November 1987.

[HUM87a] Humphrey, W. S., and D. H. Kitson: *Preliminary Report on Conducting SEI-Assisted Assessments of Software Engineering Capability,* SEI Technical Report CMU/SEI-87-TR-16, DTIC: ADA 183429, July 1987.

[HUM87b] Humphrey, W. S., and W. L. Sweet, *A Method for Assessing the Software Engineering Capability of Contractors,* SEI Technical Report CMU/SEI-87-TR-23, DTIC: ADA 187230, September 1987.

[HUM89a] Humphrey, W. S., D. H. Kitson, and T. C. Kasse, *The State of Software Engineering Practice: A Preliminary Report,* SEI Technical Report CMU/SEI-89-TR-1, DTIC: ADA 206573, February 1989.

[HUM89b] Humphrey, W. S., *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.

[LEM88] Lempp, P., "Productivity and Quality Gains Reported by Users of EPOS the Integrated CASE Environment," *Electro/88 Conference Record*, Boston, Mass., May 10-12, 1988.

[PER68] Naur, P., and B. Randell, "Software Engineering," NATO Science Committee, October 1968.

[STE87] Stenning, V., "On the Role of an Environment," *9th International Conference on Software Engineering*, Monterey, Calif., March 30, 1987.